

RabbitQR: Fast and flexible Big Data Processing at LSST data rates using existing, shared-use hardware

Ralf Kotulla^a, Arvind Gopu^b, and Soichi Hayashi^b

^aDepartment of Astronomy, University of Wisconsin - Madison, 475 N Charter St, Madison, WI, 53706, USA

^bPervasive Technology Institute, Indiana University, 2709 E 10th St., Bloomington, IN, 47408, USA

ABSTRACT

Processing astronomical data to science readiness was and remains a challenge, in particular in the case of multi detector instruments such as wide-field imagers. One such instrument, the WIYN One Degree Imager, is available to the astronomical community at large, and, in order to be scientifically useful to its varied user community on a short timescale, provides its users fully calibrated data in addition to the underlying raw data. However, time-efficient re-processing of the often large datasets with improved calibration data and/or software requires more than just a large number of CPU-cores and disk space. This is particularly relevant if all computing resources are general purpose and shared with a large number of users in a typical university setup. Our approach to address this challenge is a flexible framework, combining the best of both high performance (large number of nodes, internal communication) and high throughput (flexible/variable number of nodes, no dedicated hardware) computing. Based on the Advanced Message Queuing Protocol, we have developed a Server-Manager-Worker framework. In addition to the server directing the workflow and the worker executing the actual work, the manager maintains a list of available worker, adds and/or removes individual workers from the worker pool, and re-assigns worker to different tasks. This provides the flexibility of optimizing the worker pool to the current task and workload, improves load balancing, and makes the most efficient use of the available resources. We present performance benchmarks and scaling tests, showing that, today and using existing, commodity shared-use hardware we can process data with data throughputs (including data reduction and calibration) approaching that expected in the early 2020s for future observatories such as the Large Synoptic Survey Telescope.

Keywords: Manuscript format, template, SPIE Proceedings, LaTeX

1. INTRODUCTION AND MOTIVATION

Modern astronomical instruments offer increased sensitivities and capabilities compared to previous generations, but often at the cost of greater data volumes and data complexity. Reducing this data in an optimal fashion therefore benefits from intimate knowledge of the entire instrument, leading to sophisticated data processing pipeline products to codify this knowledge base.

Actual data processing can be done either distributed or centralized. Centralized computing can be done on-the-fly once the user requests a given dataset (see, e.g., the Hubble Space Telescope data archive) or in preparation of a data release (e.g. the Sloan Digital Sky Survey or many other surveys). The traditional, distributed approach requires each user to download or otherwise transfers all necessary data, both science and calibration products, to a particular computer to perform data processing locally. While the latter approach has its advantages as it provides each user the ability to fine-tune the data processing to the specific needs of the dataset at hand, it quickly becomes burdensome in cases where large amounts of data need to be stored and processed, and/or the processing is compute-intensive.

Further author information: RK: kotulla@wisc.edu; AG: agopu@iu.edu

1.1 Advantages of centralized data processing

There are several important factors in favor of centralized data processing: Co-location of data storage and computational capabilities make lengthy and comparably slow data transfers unnecessary; Experienced pipeline operators, human or computer, are better equipped with the required heuristics and experience to select the best data processing strategy and calibration products for a given science goals; Single point data processing ensures data homogeneity and provides superior data and software provenance, further enhancing the scientific quality of the calibrated data products.

In addition to the processing of newly acquired data, it can become necessary to re-process large amounts of older data when better calibration products become available and/or an improved understanding of the instruments and its characteristics lead to improvements in the data processing prescription. Depending on the amount and complexity of the data involved in this re-processing this can be a major undertaking, requiring significant computational resources to be completed within a reasonable amount of time.

Parallel computing, i.e. spreading out the workload over multiple inter-connected computers, is the typical approach to solve this challenge. In most cases, data volumes can be broken down into independent chunks, that can be processed in parallel. The actual processing is then performed following one of two basic principles in parallel computing: High-performance computing (HPC) acquires a fixed number of compute-nodes, with a master process orchestrating the processing, handing out jobs to worker-nodes for processing and collecting the final results. On the other hand, high-throughput computing (HTC) splits the work-load before processing starts (e.g. reduce one or more frames at a time) and puts them into a work-queue to be executed by independent workers. Both paradigms have their advantages and disadvantages: High-performance computing allows to design more complex, multi-stage processes to execute multiple processing steps in sequence, but typically requires a fixed node allocation ahead of time that can not be reduced or expanded during processing to match the required computing demands. High-throughput computing, on the other hand, easily handles a fluctuating worker node pool, but orchestrating multiple reduction steps is more challenging.

1.2 Framework for large-scale data re-processing

Here we present our implementation of a centralized processing framework for the One Degree Imager (ODI, see [1]) installed at the 3.5 m WIYN telescope located at the U.S. Kitt Peak National Observatory near Tucson, Arizona that combines some of the advantages of both high-performance *and* high-throughput computing in that it allows the quality of work orchestration and inter-node communication found in high-performance computing with the flexibility with respect to number of worker nodes typical of high-throughput computing. We demonstrate the power of this approach via benchmarks gathered from reducing large datasets of archival ODI data. Our approach, however, is more generally applicable to a wide range of possible application within and beyond astronomy.

2. IMPLEMENTATION: RABBITQR

Our implementation of the workflow during data processing consists of three separate software components tied together using the Advanced Message Queuing Protocol (AMQP), in our case RabbitMQ (combined with the QuickReduce [QR] pipeline this explains the name RabbitQR). The main component, handling all user-defined process orchestration is the *Server*. The server defines the workflow, initializes and defines the functionality of all workers, and also generates most of the status information back to the user. *Workers* are started independently on each compute node, much like in the case of HTP. However, unlike other cases, after startup does not directly start working independently, but rather announces its presence and availability to a third component that we call the *Manager*, whose main tasks are to keep track of all available workers and instruct new workers on how to communicate with the server.

The general communication workflow is illustrated in Figure 1 and structured as follows:

1. The first component to be started is the Manager to allow it to keep track of all workers available for processing.
2. One by one, the workers start up, and register their presence with the Manager.

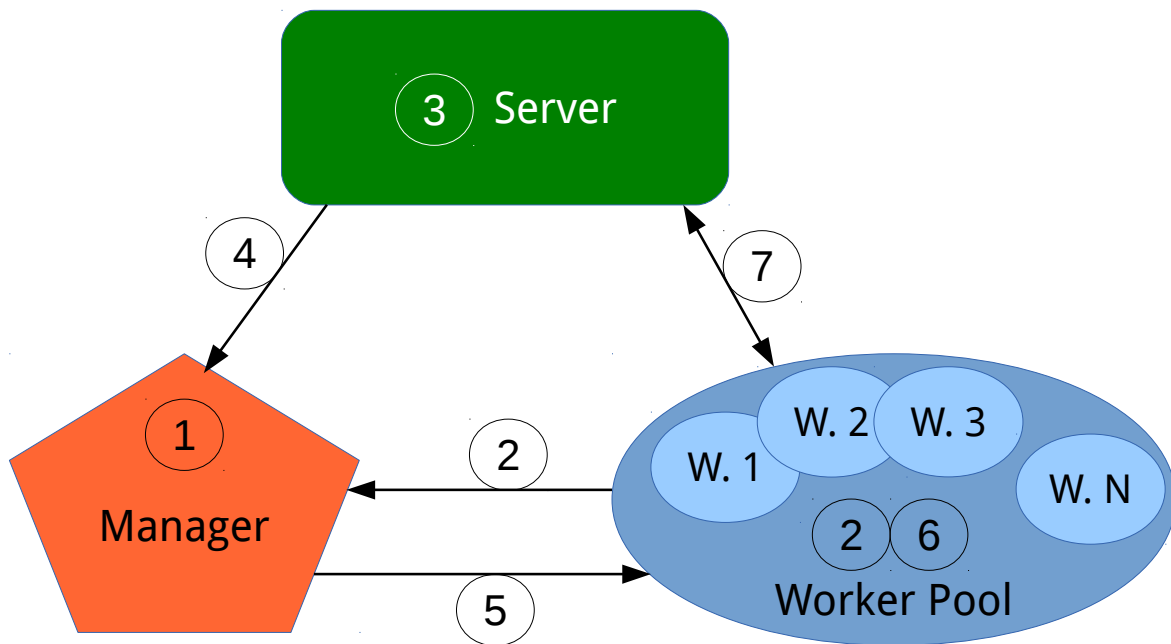


Figure 1. Illustration of the general workflow. Numbers in circles correspond to the sequence of steps described in the text in Section 2.

3. Finally, the server is started, and is processing all user-defined work-related input parameters (e.g. input files to be processed, available calibration products, etc).
4. After startup, the server requests from the manager an allocation of workers (either limited to a certain number or all available), and sends an initial setup request that contains information pertaining to the workflow at hand. All setup requests are also maintained by the Manager to allow additional workers to join the worker pool without requiring interaction by the server (see step 9).
5. The manager relies this setup request to some or all available workers.
6. The workers react to the setup request by a) subscribing to one or more message queues provided by the AMQP server, b) starting a set of server-defined processing threads to perform the actual work.
7. From now on, workers can then receive jobs from and return output and/or status information back to the server, similar to what is established during an ordinary startup of an HPC job. Once each job request has been completed, the worker properly acknowledges the receipt of the corresponding job message, signaling the AMQP server that the message has been delivered and fully acted upon. Optionally and depending on its configuration, the worker also sends a message, typically containing some output, status update, or return code, back to the server.
8. If, for some reason, a worker is removed from the worker pool during execution, all unfinished job requests received by this worker remain unacknowledged. After some per-defined timeout, the AMQP server considers the job request message as undelivered and re-inserts it into the message queue where it will be picked up by a different worker.
9. Similarly, if a worker is added to the pool after the server sent its setup request (i.e. during ongoing processing), it reports its presence to the manager (as in 2), receives its startup command (5), connects

to all relevant queues (6) and starts processing work (7) without any further need for interaction with the server.

This three-pronged approach thus provides the inter-node communication similar to an HPC job, but with the added flexibility of a variable and flexible worker pool as in HTC jobs.

An important contributor to the power of the approach presented here is that compute nodes are not required to be a homogeneous set of resources. Rather, any combination of resources (assuming they are capable of handling the processing) can be combined into the compute pool. Based on the complexity of the processing being done this can range from one to a few dedicated and/or standalone machines, to computers from local computer labs that often run idle after hours, to allocations from shared supercomputers (as in our benchmark, see below), up to machines located in the cloud (e.g. Amazon EC2, Google Compute Cloud). When combined with bundling all software requirements into a self-contained container such as Docker (see also [2] and [3]), deployment is made much easier, and flexibility with respect to use of compute resources enhanced even further.

All code for the processing framework, along with the example code used for the benchmark detailed below, will be made publicly available at https://github.com/rkotulla/distributed_processing once it is better documented to be more readily useful to others.

3. TESTING AND PERFORMANCE EVALUATION

3.1 Data set

The dataset we used as a test-case and for benchmarking achievable processing throughputs was data obtained using WIYN's flagship imager, the One Degree Imager (ODI). In short, ODI is a wide-field imager comprising, originally 13, and since its upgrade in mid-2015, 30 Orthogonal Transfer Arrays (OTAs) consisting of 64 CCDs each. Each CCD or cell is a largely independent detector and thus needs to be treated as such, requiring, amongst others, individual overscan level, gain, non-linearity factor corrections. Each OTA samples an area of $\sim 8 \times 8$ arcminutes on the sky with $4K \times 4K$ pixels with a pixel scale of $0.11''/\text{pixel}$. As test dataset for the performance benchmark detailed below we selected a random sample of ~ 500 frames taken in the ODI 5×6 configuration, consisting of 30 OTAs in a 5×6 layout, and with a total of 480 MegaPixels each.

3.2 Data processing pipeline: QuickReduce

For our demonstration case we used the latest version of the QuickReduce data reduction pipeline. QuickReduce (QR; see [4] and Kotulla 2016, in prep) is entirely python-based and fully parallelized to make use of multiple CPU cores. QuickReduce processes all OTAs independent of each other, with minimal communication to determine global reduction parameters, e.g. for sky-level estimation or fringe template scaling. Astrometric and photometric calibration are also largely parallel, with few short single-threaded parts again to compute global parameters applicable to all OTAs across the entire focal plane. For our benchmark, we apply nonlinearity, bias, dark, and flat-field corrections and apply astrometric and photometric calibration by comparing source positions and photometry to reference catalogs from 2MASS and SDSS. QuickReduce is already highly optimized towards optimal performance: Raw and calibration data (e.g. bias frames) are pre-staged to RAM-disk, as are intermediate products for use by SourceExtractor ([5]). All actual processing steps are otherwise handled entirely in memory to minimize slow I/O, in particular when using shared, network-connected filesystems.

3.3 How to access reduced data taken with ODI

In addition to the framework presented here, there are a number of ways for users to gain access to reduced and calibrated ODI data: The generally recommended way is via the ODI Portal, Pipeline, and Archive (PPA, [6]). PPA provides operator-generated data products that are fully vetted and controlled to make sure the highest-quality are available. For more exotic cases where a non-standard processing is required, PPA also gives users the option to use the QuickReduce pipeline with custom parameters and calibration products to generated tailored data products. The QuickReduce pipeline is furthermore fully integrated into the observing interface at the telescope, providing fully calibrated data products (but using pre-generated calibration products) in near real time, typically within 30 seconds after data acquisition. Additional efforts to bring data processing capabilities

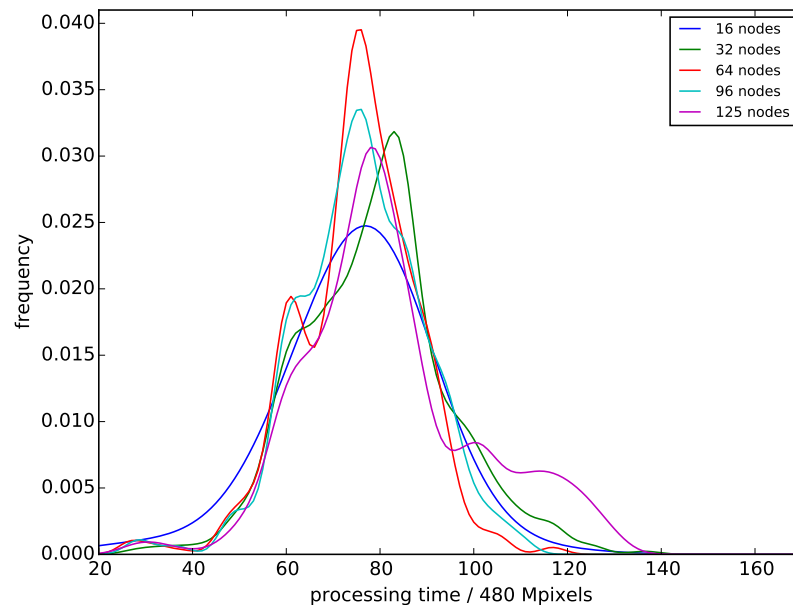


Figure 2. Distribution of processing times for the same sample of data frames, processed with different numbers of compute nodes.

to the data are described in detail in Gopu et al and Young et al, in this proceeding. Finally, users can choose the old-fashioned approach of downloading raw data and software and processing data on their own computers, but due to limited support capabilities from WIYN and demands imposed by the data processing itself (e.g. disk-storage) this data processing strategy should only be considered as last resort.

3.4 Computational resources and setup

All benchmarks were executed on the Big Red II supercomputer hosted at Indiana University, using a temporary allocation of 128 nodes. Input data was read from and output written to the Data Capacitor 2 distributed, Lustre-based, file-system. We point out that Big Red II is a shared computational resource, used by a large number of diverse users, thus causing varying loads in particular with respect to I/O on the shared file-system. Each compute node contained 2 CPUs with 16 CPU-cores each, and 64 GB of RAM. Full details and specifications of the infrastructure can be found at <https://kb.iu.edu/d/bcqt>.

4. BENCHMARK RESULTS

For our performance benchmark, we processed the same input data set with a varying number of compute nodes to test the overall performance, scalability, and total achieved throughput. The key measure for performance here is the time spent on processing each frame, measured from the moment QR was started until the final output file was completely written (not that due to OS-level caching this might not always coincide with the time the data is actually written to disk, although the additional time incurred is likely small and negligible compared to the overall processing time).

Figure 2 shows the distribution of processing times for each node-configuration, from a minimum worker pool size of 16 nodes to the maximum worker pool size of 125 nodes (for ease of use, we used dedicated nodes for both Server and Manager). Overall we find that processing times do not depend strongly on the number of nodes, with a median processing time of ~ 80 seconds for each of the ODI frames (of 480 MegaPixels each). Only at the largest node configuration with 125 workers do we find a small extension of the processing time to longer times ($\sim 100 - 130$ seconds), possibly due to the increased load on the file-system during the writing of the output files – a rough estimate of $125 \text{ (nodes)} \times 1.9 \text{ GB per file every } \sim 90 \text{ seconds}$ gives a time-averaged data

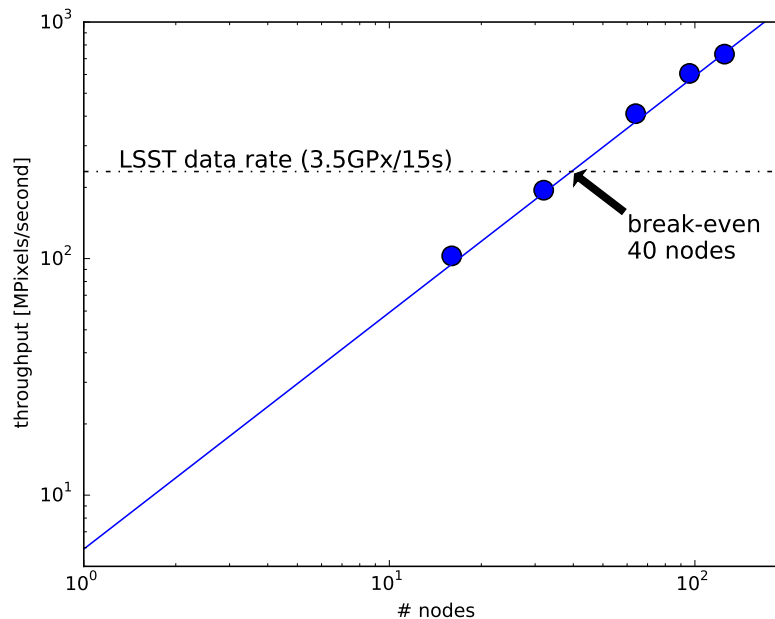


Figure 3. Scaling of integrated achieved data processing throughput as function of number of compute nodes. The horizontal line marks the data rate generated by LSST, i.e. one 3.5 GigaPixel image every 15 seconds.

rate of ~ 2.6 GB/s or ~ 21 GBit/s, a sizable fraction of the 40 GBit/s bandwidth of the shared filesystem (not including data I/O by other users).

Based on the timing information for each of the individual frames we can also compute the total, integrated achieved data throughput. Results, as function of the number of nodes, is shown in Figure 3. As expected from the detailed timing distribution (see Figure 2) we find a very linear scaling of integrated throughput as function of number of compute nodes, with total throughput being described by $\text{throughput} = 5.9 \text{ MPix/s} \times \text{\#nodes}$, for a peak throughput (using our 128 node allocation) of 625 MPixels/second. This underlines both the power and capability of the processing frame work described in this paper, as well as the performance of the optimized QuickReduce pipeline at the heart of the processing.

We also compared the achieved throughput with data rates expected from the Large Synoptic Survey Telescope (LSST). In a nutshell, LSST will observe the sky using a final camera size of 3.5 GigaPixels with typical exposure times of 15 seconds (see [7, 8]). To keep up with the incoming data, data therefore needs to be processed at a rate of 3.5 GigaPixels every 15 seconds, or ~ 230 MPixels/s. Using our setup, we can match this data throughput with ~ 40 nodes, illustrating that data processing at data rates not expected before the end of the decade is feasible already today, and is so even using shared, multi-purpose cyberinfrastructure. We do note, however, that the data processing described here, i.e. data reduction and calibration and to some degree source detection and characterization, is only part of the larger data processing effort being done by LSST.

5. SUMMARY

Processing large amounts of data fast and efficiently, both in terms of computing time as well as development time, is becoming increasingly important as new instrumentation producing ever increasing data volumes are being developed. Here we present a new framework facilitating high-performance computing distributing a workload over a potentially large and flexible number of worker nodes tied together via AMQP. The benefit of this approach is that two of the three main components (Manager and Worker) require no adaptation for different applications and thus can run on any number of facilities, from local desktop machines to cloud-based compute hosts. All workflow management, including worker setup, is contained entirely within the Server component, making development and adaptation comparably fast and easy.

Combining this processing framework with the highly optimized QuickReduce pipeline developed for the WIYN One Degree Imager we can, already today, reduce and calibrate data on a shared compute cluster within a university setting with integrated throughputs exceeding that expected from LSST in a few years.

REFERENCES

- [1] Harbeck, D. R., Boroson, T., Lesser, M., Rajagopal, J., Yeatts, A., Corson, C., Liu, W., Dell’Antonio, I., Kotulla, R., Ouellette, D., Hooper, E., Smith, M., Bredthauer, R., Martin, P., Muller, G., Knezek, P., and Huntten, M., “The wiyn one degree imager 2014: performance of the partially populated focal plane and instrument upgrade path,” (2014).
- [2] Gopu, A., Hayashi, S., Young, M. D., Kotulla, R., Henschel, R., and Harbeck, D. R., “Trident: Scalable compute archive, and visualuzation/analysis systems,” (2016).
- [3] Young, M. D., Hayashi, S., Gopu, A., and Kotulla, R., “Stardock: Shipping customized computing environments to the data,” (2016).
- [4] Kotulla, R., “The QuickReduce Data Reduction Pipeline for the WIYN One Degree Imager,” in [*Astronomical Data Analysis Software and Systems XXIII*], Manset, N. and Forshay, P., eds., *Astronomical Society of the Pacific Conference Series* **485**, 375 (May 2014).
- [5] Bertin, E. and Arnouts, S., “SExtractor: Software for source extraction.,” *AAPS* **117**, 393–404 (June 1996).
- [6] Gopu, A., Hayashi, S., Young, M. D., Harbeck, D. R., Boroson, T., Liu, W., Kotulla, R., Shaw, R., Henschel, R., Rajagopal, J., Stobie, E., Knezek, P., Martin, R. P., and Archbold, K., “Odi - portal, pipeline, and archive (odi-ppa): a web-based astronomical compute archive, visualization, and analysis service,” (2014).
- [7] Kahn, S. M., Kurita, N., Gilmore, K., Nordby, M., O’Connor, P., Schindler, R., Oliver, J., Van Berg, R., Olivier, S., Riot, V., Antilogus, P., Schalk, T., Huffer, M., Bowden, G., Singal, J., and Foss, M., “Design and development of the 3.2 gigapixel camera for the Large Synoptic Survey Telescope,” in [*Ground-based and Airborne Instrumentation for Astronomy III*], McLean, I. S., Ramsay, S. K., and Takami, H., eds., *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series* **7735**, 0 (2010).
- [8] Abell, P. A., Allison, J., Anderson, S. F., Andrew, J. R., Angel, J. R. P., Armus, L., Arnett, D., Asztalos, S. J., Axelrod, T. S., Bailey, S., et al., “Lsst science book, version 2.0,” (2009).